1989-06

A computer simulation study of mission planning and control for the NPS autonomous underwater vehicle

Nordman, Douglas B.

Monterey, California. Naval Postgraduate School

NPS Technical Report Number NPS52-89-035

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A COMPUTER SIMULATION STUDY OF MISSION PLANNING AND CONTROL FOR THE NPS AUTONOMOUS UNDERWATER VEHICLE

by

Douglas B. Nordman

June 1989

Thesis Advisor: Professor Robert B. McGhee

Prepared for:

Naval Postgraduate School
Monterey, CA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. C. Austin
Superintendent

Harrison Shull
Provost

This thesis is prepared in conjunction with research funded by the Naval Postgraduate School under the cognizance of the Naval Surface Warfare Center, White Oak.

This thesis is issued as a technical report with the concurrence of:

UNCLASSIFIED
CURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
b DECLASSIFICATION/DOWNGRADING SCHEDULE	
PERFORMING ORGANIZATION REPORT NUMBER(S) NPS52-89-035	5. MONITORING ORGANIZATION REPORT NUMBER(S)
a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b. OFFICE SYMBOL (If applicable) Code 52Mz
7a. NAME OF MONITORING ORGANIZATION Naval Surface Warfare Center (White Oak Det.)	
c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	7b. ADDRESS (City, State, and ZIP Code) Silver Spring, MD 20903-5000
a. NAME OF FUNDING/SPONSORING ORGANIZATION Naval Postgraduate School	8b. OFFICE SYMBOL (If applicable) Code 52Mz
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER O & MN, Direct Funding	
c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.

1. TITLE (Include Security Classification)
A COMPUTER SIMULATION STUDY OF MISSION PLANNING AND CONTROL FOR THE NPS AUTONOMOUS UNDERWATER VEHICLE

2. PERSONAL AUTHOR(S)
NORDMAN, Douglas B.

3a. TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1989 June	88

6. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

7. COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Autonomous underwater vehicles, artificial intelligence, robotics, graphics.

9 ABSTRACT (Continue on reverse if necessary and identify by block number)

Autonomous vehicles will operate where humans cannot or do not want to go. The last decade's advances in computer processor capability and speed, component miniaturization, signal processing, and high-energy-density power supplies have made remotely-operated vehicles (ROVs) a reality. These reliable, long-range, high-endurance vehicles now perform a number of tasks in research, industrial, and military applications, but they are still incapable of truly autonomous behavior.

The U.S. Navy has identified a number of autonomous vehicle missions, and the Naval Postgraduate School is extending ROV technology to build an autonomous underwater vehicle (AUV). The mission controller for the NPS AUV is a knowledge-based artificial intelligence (AI) system requiring thorough analysis and testing before the AUV is operational. Rapid prototyping of this software has been demonstrated by developing controller code on a LISP machine and using an Ethernet link with a graphics workstation

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Professor Robert B. McGhee	(408) 646-2449	Code 52Mz

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

Block 19.

controller's environment. This thesis updates and improves the earlier simulator and its hardware, and describes the development of a new testing simulator design to examine AUV controller subsystems and vehicle models before integrating them with the full AUV for its test environment missions. This AUV simulator is fully autonomous once initial mission parameters are selected.

Approved for public release; distribution is unlimited.

A COMPUTER SIMULATION STUDY OF MISSION PLANNING AND CONTROL FOR THE NPS AUTONOMOUS UNDERWATER VEHICLE

by

Douglas B. Nordman
Lieutenant, United States Navy
B.S., United States Naval Academy, 1982

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1989

ABSTRACT

Autonomous vehicles will operate where humans cannot or do not want to go. The last decade's advances in computer processor capability and speed, component miniaturization, signal processing, and high-energy-density power supplies have made remotely-operated vehicles (ROVs) a reality. These reliable, long-range, high-endurance vehicles now perform a number of tasks in research, industrial, and military applications, but they are still incapable of truly autonomous behavior.

The U.S. Navy has identified a number of autonomous vehicle missions, and the Naval Postgraduate School is extending ROV technology to build an autonomous underwater vehicle (AUV). The mission controller for the NPS AUV is a knowledge-based artificial intelligence (AI) system requiring thorough analysis and testing before the AUV is operational. Rapid prototyping of this software has been demonstrated by developing controller code on a LISP machine and using an Ethernet link with a graphics workstation to simulate the controller's environment. This thesis updates and improves the earlier simulator and its hardware, and describes the development of a new testing simulator designed to examine AUV controller subsystems and vehicle models before integrating them with the full AUV for its test environment missions. This AUV simulator is fully autonomous once initial mission parameters are selected.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

The code for the original AUV simulator was provided by Dave Macpherson and was modified to conform to new hardware and different operating systems. The NPS AUV simulator was developed for a graphics project by Dave Marco, Ray Rogers, and Mike Schwartz. The network communications software that ties all the machines together was written by Ted Barrow, Sehung Kwak, Bill Teter, and Larry Shannon.

The assistance of Albert Wong of the NPS Computer Science Department Technical Support Staff was crucial in acquiring the required operating systems and invaluable in getting them installed and operating.

I am especially grateful to Professor Sehung Kwak for his assistance to me based on his intimate knowledge of Lisp machines and their network software.

I. INTRODUCTION

A. BACKGROUND AND BRIEF PROBLEM STATEMENT

Autonomous vehicles can go where humans cannot or do not want to go. These robots are capable of receiving initial input, moving to another location and executing a mission, and returning with the requested results or data. In addition to performing labor-intensive or repetitive tasks, these vehicles can perform their jobs faster and with greater precision than humans, and can also proceed into hostile or contaminated environments.

For the last thirty years, remotely-operated vehicles (ROVs) have attempted to fill these needs. The last decade's tremendous advances in computer and systems engineering have produced powerful, reliable, and inexpensive ROVs capable of a wide variety of tasks. These vehicles are commonplace in the oil-drilling, salvage, and ocean engineering industries, and they are extending the reach of oceanographers with lengthy missions that produce high-resolution data from great depths. The armed forces of several countries use ROVs for battlefield reconnaissance and long-range targeting, and military ROV research continues. (Bane and Ferguson, 1987.)

Although they have become extremely useful to the military, ROVs still require operator supervision and are incapable of independent operations. This handicap greatly reduces their ability to execute complicated, covert missions in hostile territory where an unknown and dangerous environment requires impromptu

planning to handle unforseen situations. Since ROV systems have proven their military usefulness, future designs must make these combat vehicles truly autonomous.

The U.S. Navy has identified a number of tasks that can be performed by Autonomous Underwater Vehicles (AUVs), and the Defense Advanced Research Projects Agency (DARPA) strongly supports AUV research (Robinson, 1986; Eisenstadt, 1987). Researchers are applying ROV technology to design powerful AUV systems with high endurance, small profiles, and extended range. However, the greatest challenges lie in producing artificial intelligence (AI) systems to support mission execution, vehicle perception and navigation, and contingency planning.

The Naval Postgraduate School is developing an experimental AUV to address these military requirements. Part of this project is the design of simulators that will reduce the time and expense of implementing various AUV subsystems while also permitting efforts to proceed along several simultaneous approaches. Previous simulator research (MacPherson, 1988) has shown that graphics workstations provide a useful way to simulate a realistic environment for conducting AUV operations. This approach permits the prompt development and thorough testing of AI software and will be used to test code for the NPS AUV while also developing the next generation of software.

This thesis improves on the original research by expanding and upgrading its systems. In addition, a new simulator has been developed to generate a "laboratory environment" for testing several AUV planning, navigation, and control subsystems. This new simulator will also be used to examine different AUV hydrodynamic

models and to test maneuvering systems in conjunction with different sensor configurations. These proven subsystems will be integrated with the AUV mission planner to develop full-scale proposed missions before operational testing of the actual vehicle.

B. THESIS ORGANIZATION

Chapter II reviews previous work on ROV and AUV systems and examines system architecture and programming language issues. The use of modern computer workstations for vehicle simulations is discussed and an example is included.

Chapter III presents a detailed problem statement for this thesis and describes the mathematical model for the dynamics of both vehicles. The autopilot and logical control levels are discussed and the features of the two vehicle models and their environments are compared. The simulation facilities are also described and contrasted.

A detailed description of the simulator's design and operation is presented in Chapter IV. This includes an examination of the autopilot, the dynamics of both vehicles, the operation of the inter-computer communications code, and a user's manual. The overall software system design is described to show how actual vehicle changes or improvements can be added to the simulator to keep it as realistic as possible.

The simulator's operation is examined in Chapter V. This chapter explains the various missions that can be run on the simulator and their results. These results are summarized in Chapter VI and are used as a basis for proposed

extensions and improvements. This chapter also explains the contributions this research has made to the development of autonomous vehicles.

II. SURVEY OF PREVIOUS WORK

A. INTRODUCTION

Remotely-operated vehicles (ROVs) have been in operation since the early 1950s and autonomous vehicle research began in the early 1960s. Although the early ROVs were limited in range, notoriously difficult to control, and mechanically unreliable, autonomous vehicles were even more severely restricted by available technology. Through the early 1970s, manned vehicles were required for most tasks while maturing ROV technology developed useful ranges, better controls, and reliability. However, although ROV use began to grow, technological handicaps caused autonomous vehicle research to die out by the end of the 1960s. (Brady et al., 1984.)

The 1980s have produced dramatic improvements in processor capability and speed, component miniaturization, signal processing, sensor resolution, and high-energy-density power supplies. These rapid advances have been integrated with ROV operations and have rekindled autonomous vehicle research for a variety of military and industrial uses. ROV applications are widespread in the military and have become commonplace in many industries. In 1986, the Defense Advanced Research Projects Agency (DARPA) AUV Study Committee identified over seventy military AUV missions (Bane and Ferguson, 1987) while additional groups described many other military and industrial uses for AUVs (Robinson, 1986). U.S. Navy ship designers are also accommodating AUV technology-- the *Sea Wolf* (SSN-21)

class of attack submarines will have larger-diameter torpedo tubes capable of deploying AUVs for anti-submarine warfare or other uses (Baker, 1989).

This chapter surveys current ROV uses and then shows how this technology and operational experience has been applied to AUV research. Different AUV programs are described and their research issues are discussed. The last section of this chapter discusses the use of computer simulations for the research and testing of vehicle dynamics and controls.

B. REMOTELY-OPERATED VEHICLE (ROV) TECHNOLOGY

MacPherson (1988) describes several military ROV programs under development or in operation. The U.S. Navy's Remotely-piloted Vehicle (RPV) for airborne ocean surveillance and the Israeli Armed Forces battlefield RPVs have seen several years of operating experience; in some cases their utility has been demonstrated in actual combat (Steele, 1988).

In contrast to the military's recent ROV experience, industrial ROVs have undergone explosive growth in the last five years. Researchers from industry and academia have formed many groups, and several annual symposia display a broad range of ROVs for a variety of applications and budgets (ROV '89, 1989). These products are frequently used in the offshore oil-drilling and underwater construction fields where the danger or expense of using human workers is prohibitive. Another well-publicized operation was the use of the *Alvin* and its *Jason Jr.* ROV to locate and explore the *Titanic* (WHOI, 1986; Yoerger et al., 1986). In addition to the salvage industry, ROVs have gained widespread acceptance in oceanographic

research. The remainder of this section will discuss several of these programs and their ROVs.

1. *Sea Ferret*

The Underwater Resources, Inc. *Sea Ferret* (or "miniROVer") is a small, low-cost ROV used in tunnels and penstocks to perform corrosion surveys and welding inspections. This vehicle operates in the tunnels of hydroelectric plants or water-treatment facilities to observe valve or gateway operations, perform damage assessments, make repairs, and perform non-destructive weld testing. Although it appears to be an expensive "high tech" application for "low-tech" facilities, the *Sea Ferret* has saved millions of dollars in maintenance, troubleshooting, and emergency repair costs while reducing the manpower and risks associated with this type of work. (Underwater Resources, 1989.)

2. Sea Owl

The *Sea Owl* is a small industrial ROV from Scandinavian Underwater Technologies (SUTEC) that incorporates many of the latest design concepts in the ROV industry. Instead of a torpedo-shaped body-of-revolution hull or an open-frame cage to house its components, the *Sea Owl* uses a hydrofoil twin-cylinder hull that the operator "flies" through the water. While earlier ROVs used a single propulsor, the *Sea Owl*'s seven thrusters give the vehicle maneuvering agility and mechanical redundancy. With its five-foot length, two-foot beam, three-knot speed, and 1500-foot depth limit, this ROV is a low-cost system used by oil platforms and salvage operations. The "300" version has many different near-shore applications

and the larger 500 model is a more expensive platform suitable for open-ocean salvage or exploration. (SUTEC, 1989.)

3. *Sea Twin*

The *Sea Twin* (also by SUTEC) is a follow-on to the *Sea Owl* design that adds extra features for a different environment. With its larger hull and additional thruster, this ROV is nearly fifty percent bigger and has twice the displacement of the *Sea Owl*. The greater size and thrust is used for higher power and increased stability in rapid currents where smaller ROVs are unable to operate. In addition to its vehicle improvements, the *Sea Owl* system also uses fiber-optic signal transmission technology to produce a smaller, low-drag tether cable with a much wider signal bandwidth. This advanced cable makes the ROV capable of simultaneously handling several different types of sensors or of transmitting real-time high-resolution video. (SUTEC, 1989.)

4. ECA

This French corporation is one of the world's largest ROV producers and is noted for their low-cost designs of highly adaptable vehicles. Their *Pope* ROV is a general-purpose vehicle used for near-shore or coastal operations, with a variant for open-ocean work. Another ROV, the *PAP 104*, is designed for high-resolution searches and charting applications; its manipulators also possess a high degree of manual dexterity that make this system very effective in underwater demolition. The *PAP 104 Mk5* is a military version used for mine warfare. As of early 1989, ECA has sold over 325 ROVs to many corporations and to the navies of eleven European and Asian countries. (ECA, 1989.)

5. *Gemini 6000*

Eastport International, Inc., has made significant advances in the use of ROVs for open-ocean search and salvage. Their *Gemini 6000* ROV is the size of a small auto with a 9,000 pound displacement and two seven-function hydraulic manipulators. The vehicle uses several photographic systems and a fiber-optic link to transmit real-time high-resolution color video as well as record 35mm pictures and stereo photographs.

In 1985, an earlier version of the *Gemini 6000* located and recovered most of the remains of a Boeing 747 aircraft from a depth of 7000 feet. In 1986, this ROV mapped the crash site of the space shuttle *Challenger*, and the vehicle augmented diver's salvage efforts by retrieving many of the spacecraft's fragments from deeper waters. (SUBNOTES, 1989a.)

In February 1989, the *Gemini 6000* recovered an aircraft flight recorder from a new record salvage depth of 14,800 feet. This was also the first use of an ROV in an aircraft accident investigation; it involved a survey of more than 200 objects in an area of over a square mile. During 345 hours of bottom time, the *Gemini 6000* transmitted many hours of high-resolution video, took over 2400 35mm photos, and recorded over 300 stereoscopic photos. (Sea Technology, 1989.)

6. Oceanographic Research

The Monterey Bay Aquarium Research Institute (MBARI) is a non-profit research institute founded in 1988 to study the oceanography of Monterey Bay on the central California coast. The bay is actually a deep underwater canyon that starts less than a mile offshore and reaches depths in excess of 5000 feet within ten

miles of the coast. The canyon's close proximity to land and its unusual features, the rich diversity in the bay's ecology, and a large number of oceanographic research facilities in the area have all combined to give MBARI a number of opportunities to conduct deep-ocean research.

To support this work, MBARI uses an open-frame ROV from International Submarine Engineering of Canada. The vehicle, also about the size of a small car, has four thrusters, seven camera systems, and a seven-function manipulator. It is capable of operating in depths up to 6000 feet and has already used its deep-ocean collecting, surveying, and photographic abilities to help explain several unresolved research questions. (SUBNOTES, 1989b.)

C. AUTONOMOUS UNDERWATER VEHICLE TECHNOLOGY

The technology used in ROVs is directly applicable to AUVs since both types of vehicles require sophisticated controls, rapid maneuverability, extended ranges, high-resolution sensors, and high-capacity signal processing. However, the most significant AUV technological development has been the large-scale introduction of cheap, high-performance computer processors. The wide availability of the Intel 80386 and Motorola 68030 series of processors, as well as RISC and VME architectures, has moved AUV projects out of the lab and into industrial development (SUBNOTES, 1989). Research has produced ROVs that routinely operate in the harsh ocean environment; the next challenge is to make autonomous vehicles that are as reliable as ROVs and that will perform tasks which ROVs cannot. This section will survey industrial, academic, and military AUV research.

1. *XP-21*

Applied Remote Technology, Inc. (ART), has developed a test hull for its prototype AUV. The *XP-21* has a 16-foot torpedo-shaped hull 21 inches in diameter with a displacement of 1700 pounds; it can accommodate a 700-pound payload of up to eight cubic feet. This vehicle can submerge to 2000 feet with a six-hour endurance at six knots. ART is using the *XP-21* to verify the performance of hardware and software for navigation, guidance, control, and communications subsystems that will be incorporated into a larger AUV. (SUBNOTES, 1988; ART, 1989.)

2. *PTEROA*

The University of Tokyo has developed a small AUV for independent sea floor mapping. The *PTEROA* is roughly five feet long with a three-foot beam and a maximum speed of 3.5 knots on two small thrusters. The designers deliberately simplified the AUV's control and propulsion systems to minimize power demands and to reduce computing overhead. The vehicle submerges with ballast to allow it to descend in a slow glide to the sea floor. Once near the bottom, the *PTEROA* drops its ballast and proceeds along its search pattern, recording data while using its sonar to navigate along bottom contours. (Tamaki, 1989.)

3. EAVE East

One of the most advanced operational AUV research programs is conducted by the University of New Hampshire Marine Systems Engineering Laboratory. The Experimental Autonomous Vehicle (EAVE) is an open-frame, highly-maneuverable submersible with two models for use as system development

test beds. (This program is called "EAVE East" to distinguish it from the San Diego Naval Ocean Systems Center's "EAVE West" tethered vehicle.)

Where feasible, the EAVE East uses redundant sensors to verify its inputs for critical reliability in a hostile operating environment. A pressure transducer and a fathometer measure the vehicle's depth while a five-beam sonar is used for obstacle detection and avoidance. The EAVE's navigation module first fixes its position with both long- and short-baseline acoustic systems and then verifies the AUV's heading with an on-board magnetic compass. (Jalbert, 1987.)

The EAVE's computer architecture achieves an innovative integration of several different hardware and software systems. A VME bus coordinates eight Motorola 68000-series processors with 4 Mbytes of RAM and an 800 Mbyte optical disk. Several sub-processors handle the tasks of external I/O, rapid memory access, and memory caching without lowering execution speed. A real-time operating system supports the AUV's Portable Common LISP Subset (PCLS) software. This operating system is used to divide the EAVE's resources into a high-level and a low-level hierarchical architecture for mission planning and vehicle control.

The high-level component of the system uses five of its processors to update and evaluate the environmental knowledge base, to replan the mission if unexpected events or casualties occur, and to supervise the lower-level system components. These processes run in a loop with approximately a 100-second cycle time. The low-level portion uses the remaining three processors to analyze sensor data, control vehicle motion, and collect additional sensor inputs approximately once per second. (Shevenell, 1987.)

The vehicle's hierarchical control structure closely mimics the division of responsibility used in human chains of command. The mission-level planner is at the top of this hierarchy and accepts human user input for the mission's objectives and priorities. The planner develops a launch-to-recovery mission profile that examines energy requirements, time limits, and risk factors to evaluate a number of alternatives and select the optimal mission plan. Instructions are then sent to lower-level planners that deal only with the next specific sub-task to be performed. As the mission proceeds, the top-level planner updates its status and replans the mission to accommodate unexpected events. This structure produces the intelligent, adaptive behavior that autonomous vehicles require for executing complex missions in a hostile environment. (Blidberg and Chapell, 1986.)

4. DARPA Funding

Encouraged by the rapid development of industrial and academic AUV programs, the Defense Advanced Research Projects Agency (DARPA) has committed up to $100 million to support AUV projects with military applications (Eisenstadt, 1987). This includes a recent $23.9 million contract issued to Draper Laboratories for the development of two AUVs to serve as test vehicles for various U.S. Navy missions. These AUVs will carry instrumented systems and will execute mission packages developed by Martin Marietta (ROV News, 1989).

D. CONTROL SYSTEM SOFTWARE ARCHITECTURES

The most significant difference between AUVs and ROVs is the substitution of an artificial intelligence (AI) control system for the human operator. Although these control systems can execute instructions and process data much more quickly

and more reliably than any human, they frequently lack the flexibility and response time needed for a rapid reaction to unexpected events or casualties. Much of today's AUV research is concentrating on the development of computer architectures, languages, and operating systems that can provide this flexibility while retaining speed and reliability. Two aspects of these issues are discussed below.

1. Distributed versus hierarchical control

Many different real-time control systems attempt to improve their overall speed by having data pass through as few processors as possible. Ideally, this system would use some sort of memory to receive and store incoming data while a master processor would examine all data and immediately dispatch it to the appropriate subsystem for further processing. After manipulating the data, all co-processors would report their results as additional data, which would then be re-dispatched to the next appropriate co-processor, and so on.

These processing schemes have been grouped under the heading of "blackboard systems" to describe the way data is posted at a central location to await pickup for analysis. A blackboard system can use object-oriented programming to share information between expert subsystems without the time-consuming overhead of data transfer, and one implementation of this scheme is being used to support the development of an AUV control system. (Doty and Wachter, 1986.)

Although blackboard architectures can minimize unnecessary data transfer, current implementations have difficulty achieving satisfactory response times or flexibility. Each expert system devotes a significant overhead (hardware

interrupts) to "checking the blackboard" for arriving data, and an attempt to assign higher priorities to time-sensitive data events frequently generates unacceptable additional overhead. Another blackboard drawback is the method by which data is identified for the appropriate expert system. Emergencies, errors, and unexpected results will be posted but may never be "picked up" by an expert subsystem if the data does not fit into an identifiable category. An exception-handling mechanism can notify the system of these problems but is usually unable to correct them, so the machine's processing gradually slows down or abruptly crashes.

A hierarchical system imitates human chains of command by dividing large projects into progressively smaller tasks that are handled at lower levels. The lower processing levels accept results from even lower levels, operate on the information, and pass their results up to a higher level. The higher levels will act on this input, pass orders and data back down to the lower level, and report their results to progressively higher levels of control. In this way, an object-oriented system can manipulate small objects that are nested in larger objects; the lower objects usually can communicate only by passing their data through their parent object.

A hierarchical architecture ensures that all data is eventually handled by some process or object. However, data-sharing between adjacent modules is restricted so the same information exchange requires more handling than a blackboard system. In addition, the system's central knowledge base is available to fewer processors so more data transfers are needed to get information to the appropriate location. Although hierarchical systems use fewer interrupts for I/O

polling, lower-level processors can block the execution of a higher processor; the higher level may be idle until data is passed up to it. Finally, while a hierarchical system will handle errors and unanticipated events more easily, the division of responsibilities is crucial to prevent higher-level processors from being overloaded by routine tasks and exception-handling.

The deficiencies of each individual architecture tend to make their separate implementations unwieldy, slow, and unreliable. However, systems that combine elements of the two schemes often achieve significant improvements with few side effects. These "hybrid" architectures can use faster blackboard concepts to reduce data transfers by making information more available while their hierarchical structures ensure that exception-handling is successfully completed. (Doty and Wachter, 1986; MacPherson, 1988.)

2. Language Alternatives

a. LISP versus Prolog

Two of today's most popular artificial intelligence languages are LISP and Prolog. Conceived in the late 1950s, LISP is one of the earliest AI languages and has had over three decades of refinement and standardization. Its modular code, list-oriented structure, and low-level operators give it a speed that justifies its recognition as the "assembly language" of AI. On the other hand, Prolog is a relatively new and extremely powerful language with a rule-based structure that makes it very useful for developing the collections of rules known as "expert systems."

Each language has several deficiencies. Compared to Prolog, LISP is faster but much harder to read. Its recursive routines make programming counter-intuitive and debugging very difficult. Its many dialects have recently been standardized but this standardization is slow and still incomplete. Although LISP executes very quickly, its low-level design also makes it difficult to implement higher-level AI abstractions. In particular, LISP lacks any built-in inferencing mechanism.

Since Prolog is a high-level language, it is easier to design high-level structures. However, the language executes more slowly than LISP and Prolog requires large amounts of memory for its back-tracking execution. Although Prolog simplifies the construction and maintenance of large rule-based systems, it can be difficult to thoroughly test these system's permutations and subsequently correct unintended side-effects. Continued improvements to Prolog and its associated debuggers will improve this language's facilities.

The top-level controller of an AUV will require a system with the speed advantages of LISP yet the power and flexibility of Prolog. The construction of such a system will depend heavily on a large library of reliable mission-execution subroutines using portable code that can function as "building blocks" for the next generation of missions and systems.

b. Expert System Shells: KEE versus ART

Expert systems have been developed to make up for LISP's inferencing deficiencies; these systems are one example of an attempt to combine the best features of LISP and Prolog. KEE (Knowledge Engineering Environment)

and ART (Automated Reasoning Tool) are examples of a class of software called "expert system shells". Programmers use these software tools to build knowledge-based expert systems and to apply them to large, complex problems. These systems are especially useful in AI applications where the problems may be ill-defined or so complex that the software system must mimic the decision-making processes of human experts.

The KEE shell contains a number of features for manipulating a knowledge base and its corresponding set of rules. Part of this system is a graphical user-interface module and a set of object-oriented programming routines; these two frames provide an intuitive and powerful means of organizing and executing LISP functions. (KEE User's Manual, 1986.)

The ART shell also uses a rule-based scheme to operate on its expert system. Four different types of rules can be used to describe and manipulate a knowledge base. A powerful ART "inference engine" will compile and execute this collection of rules and data, repeatedly drawing conclusions from the knowledge base and applying these conclusions to generate more data for the knowledge base. This scheme continues until a specific goal is achieved. (ART Reference Manual, 1986.)

Both shells use a proprietary structured language to implement their rule-based systems, and both shells are able to incorporate LISP functions into their execution for the control of other systems. Although the lack of a standard rule-based language (such as Prolog) is a drawback, the AUV simulator does not use

any rules to execute its missions. Instead, the simulator uses its shell to organize and to efficiently execute the many routines that make up the expert system.

Since these shells are applicable to many situations involving complicated path-planning or scheduling problems, they are both suitable for the mission level of the AUV simulator. Compared to KEE, the ART shell has a much more powerful rule-based language, but its interfaces are not intuitive and can present problems for inexperienced users. Since the simulator uses no rules, ART's user interface is a significant disadvantage. KEE has been chosen since it provides the simplest user interface-- one that requires little system knowledge for mission execution, and one that can be quickly manipulated with a mouse.

c. Ada

When it was faced with rapidly-rising software development and maintenance costs and reduced software portability, the U.S. Department of Defense (DoD) sponsored the development of a high-level programming language that would be suitable for a wide range of applications. The Ada programming language filled these specifications and is building a large library of portable subroutines that will reduce software development, maintenance, and compatibility costs. The DOD has subsequently decreed that Ada will be the programming language for all mission-critical U.S. military projects and has trademarked the language to enforce its standardization.

Many of the guidance, navigation, and control subsystems of an AUV or ROV can be programmed in Ada. However, an AUV's top-level planner is a specialized AI construct requiring a language with the features and power of LISP or Prolog. Ada is not yet suitable for this application.

E. COMPUTER SIMULATION OF VEHICLE DYNAMICS AND CONTROL

In the last ten years, the rapid deployment of ROVs has validated vehicle and sensor designs. ROVs are readily available for uses in an ever-increasing variety of environments and applications, and their control systems and signal-processing capabilities continue to improve. Recent ROV advances have largely been technological updates of existing concepts, and industry's experience is lowering the subsequent research and development costs. Many improvements are made by inexpensive alterations of "off-the-shelf" components instead of through new designs, and the industry will mature as vehicles become cheaper and more available to the common user.

AUV design is not so advanced. While ROVs can cheaply use rapid prototyping and testing techniques, AUVs are still quite complicated and costly. The operational testing process subjects these expensive vehicles to harsh and unpredictable environments where the logistics are difficult and some AUV losses are unavoidable. While there is no substitute for operational experience, most of the AUV's engineering problems have been solved while learning how to operate ROVs. The last step is the development of a high-level mission planner that can be simply and cheaply tested without expensive logistics and unaffordable losses.

One solution to this problem is computer simulations. Through the use of vehicle simulators, AI mission planners can be developed in the laboratory and can receive data inputs from artificial (computer-generated) sources. The planner's outputs can be used to drive a simulator whose actions can be observed and interpreted to evaluate the effectiveness of the planner without having to risk the

vehicle. The same powerful computer systems that have revived AUV research can thus be used for rapid AUV prototyping and low-risk initial testing.

One of these systems was developed and tested in 1986. A Westinghouse research team, as part of a preliminary AUV design project, programmed a test system for an AUV mission planner and navigator. The inputs and outputs for this planner are handled by a simple graphics simulator that provides the AUV with information about the surrounding environment and then revises its display to show actions ordered by the AUV mission planner. The simulator generates and updates a picture that shows the planner setting up a mission, the navigator determining a path, and the AUV executing these orders while reaching its objectives and avoiding obstacles. (Schweizer and Oravec, 1986.)

F. SUMMARY

This chapter presents a survey of previous research and accomplishments that are relevant to this thesis. A listing of representative current ROV technology is followed by a discussion of AUV research and issues. Different system architectures and programming languages are presented and will be evaluated in following chapters. This chapter concludes with a brief discussion and an example of the advantages of modern computer workstations for dynamic system simulation.

III. DETAILED PROBLEM STATEMENT

A. INTRODUCTION

This thesis provides a real-time graphical simulation of a proposed AUV and facilitates the development and testing of various control algorithms. The simulator is part of a Naval Postgraduate School (NPS) research project that will design, build, and test a series of Autonomous Underwater Vehicles.

B. VEHICLE CHARACTERISTICS

The NPS Autonomous Underwater Vehicle is modelled after the Swimmer Delivery Vehicle (SDV) used for the delivery and extraction of U.S. Navy Special Warfare Teams. The actual NPS Model 2 AUV will resemble the SDV so the simulator's vehicle dynamics have been scaled to the dimensions of the AUV currently under construction.

The simulator's controller carries out AUV operations by directing its output to either of two graphical representations. The original graphics display is a simplified vehicle that models complex AUV missions in the open-ocean environment. The second graphics display uses a more sophisticated SDV hydrodynamic model in a small "test pool" to evaluate various AUV configurations and to develop the actual control algorithms that will be used by the NPS Model 2 AUV.

1. The Original Vehicle

The first vehicle simulation (MacPherson, 1988) permits mission execution without requiring a detailed implementation of AUV dynamics. The simulator represents a small manned vehicle with a control panel and a "through the periscope" display similar to that of the U.S. Navy's *Sturgeon* class attack submarine. The vehicle has a single screw and rudder and maintains continuous neutral buoyancy. Aft, sternplanes impart a hull pitch angle for large depth changes while forward-mounted bowplanes provide more precise depth control without generating a pitch angle. Although users can manually operate the AUV, the vehicle is normally under autopilot control.

The original dynamic model consists of a simple point-mass approximation governed by one acceleration equation, two rate equations, and one attitude equation. The vehicle's location and orientation is described by applying these equations at a 10-Hz rate and by setting the autopilot's control surface positions according to depth or course error. AUV speed is chosen by the autopilot and is limited by battery charge or by the onset of cavitation. Acceleration is fixed at 1 knot/sec^2 while depth and azimuth rates depend on a combination of speed and control surface angle. The vehicle's pitch angle is assigned a steady-state value determined by the AUV's speed and sternplane angle.

Although the AUV displays rigid behavior and little inertial delay, no attempt was made to model actual submarine dynamics since these would have little impact on the large-scale decisions implemented by the mission controller. This

model is a simple and effective way to display the actions and results generated by mission execution algorithms.

2. The NPS AUV

The second simulation is based on the Swimmer Delivery Vehicle's dynamics and on preliminary NPS model hydrodynamic test data (MacDonald, 1989). The hull shape is a flattened cylinder with a rounded bow and a tapered stern; the AUV maneuvers with bow planes, stern planes, twin rudders, and twin screws. The dynamics model uses a vehicle mass of 12,000 pounds at neutral buoyancy with a length of 17 feet, a beam of five feet, and a height of 2.5 feet. The NPS Model 2 AUV is equipped with two vertical and two horizontal thrusters, so the simulator image also shows these thrusters. (See Figure 3.1.)

The AUV's position, orientation, and velocity is determined by calculating hydrodynamic drag forces and Euler angle rates and then updating these parameters at a 30-Hz rate. The AUV is displayed from an external point of view instead of the earlier "through the periscope" perspective. The simulation algorithm is a considerable improvement over the original model since the AUV exhibits realistic acceleration and inertial behavior.

The original version of this simulator (Schwartz, 1989) was designed to evaluate AUV hydrodynamic coefficients and to examine the resulting vehicle dynamics under a variety of speeds and pitch angles. The simulator relies on the user's manipulation of the vehicle's control surfaces and its speed; there is no autopilot controller.

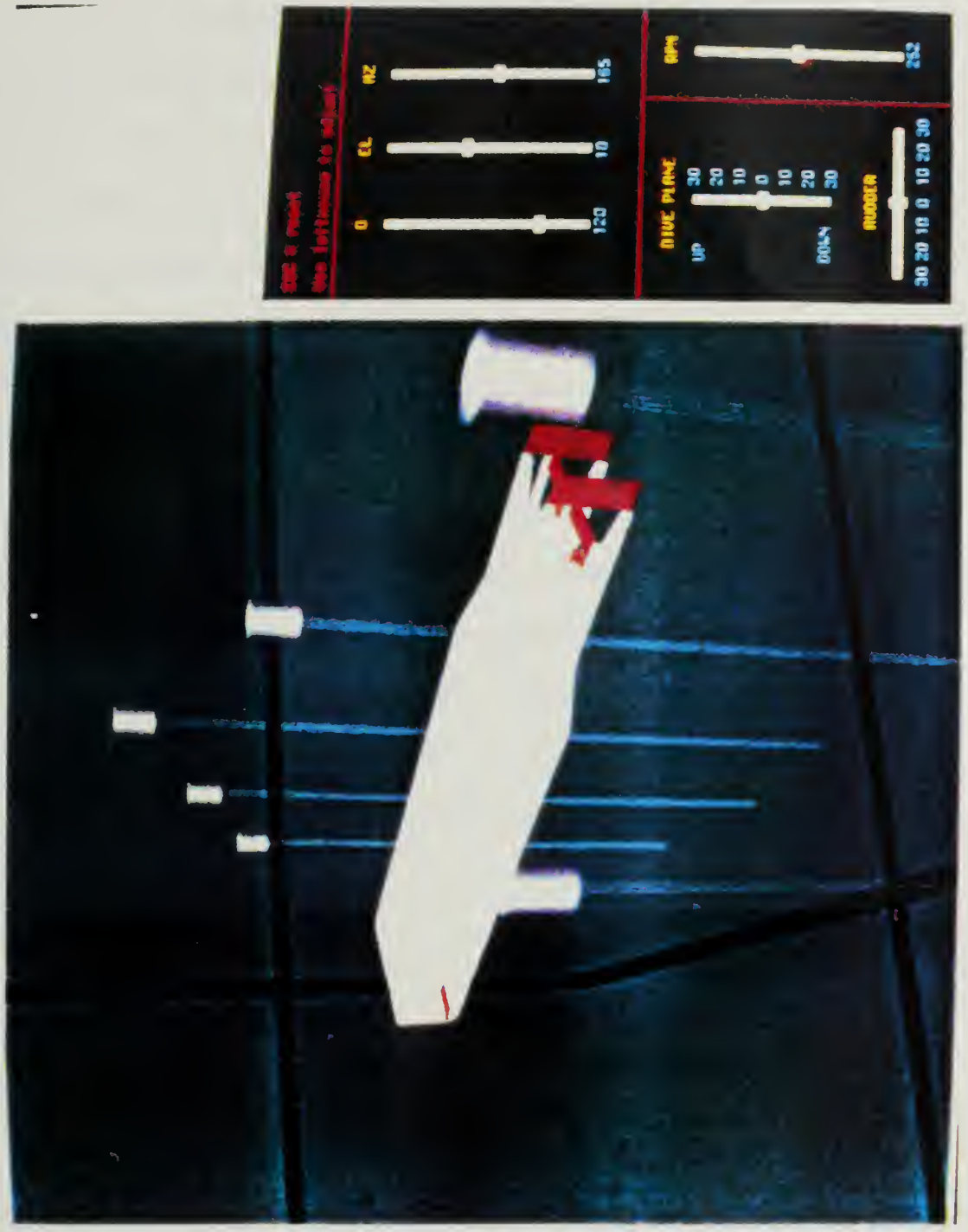

Figure 3.1 NPS AUV Simulator

The second generation of this program uses a simple autopilot depth- or course-error calculation to set control surface positions for maneuvers. Autopilot orders create control surface angles which in turn act on the AUV hydrodynamic model to generate hull pitch angles and resulting changes in depth or course. Although this first-order controller produces abrupt and non-linear control surface behavior, the NPS AUV design team is developing an advanced control system. The simulator's code structure will allow this advanced controller algorithm to be installed between the autopilot and the hydrodynamic model.

Although the hydrodynamic model has a length of 17 feet, the vehicle displayed on the simulation graphics workstation is scaled to a length of five feet. This inconsistency between the hydrodynamic model and the visual vehicle makes the simulator's NPS AUV appear to maneuver differently from the characteristics of a 17-foot model. This effect does not affect the simulator's mission planning; the inconsistency will be corrected when the hydrodynamic model of the NPS AUV is incorporated into the simulator.

C. ENVIRONMENT

The ocean environment for the original AUV simulator is described in detail in MacPherson (1988). The simulation begins with the AUV at periscope depth in a sector of water five nautical miles on a side. The sea floor of this model is a submerged cone with an exposed island (the cone's vertex) near the center of the sector. In addition to the island and its shoals, the AUV must also contend with a number of surface contacts-- military vessels, merchant ships, and buoys. The large

body of water and its congested environment provide a realistic test of the AUV's mission control and guidance software.

The water environment for the NPS AUV simulator is modelled after a proposed test site for the actual vehicle. The simulation displays a "swimming pool" (120 feet by 60 feet by eight feet deep) containing a number of submerged cylindrical obstacles. The simulator's AUV and test pool are scaled to the actual sizes of the NPS AUV and its test site, although the hydrodynamic model is not scaled to this environment. (See earlier discussion in B.2.) The test pool is a much simpler environment than that of the earlier simulation and is intended to give the AUV design team a realistic way to test various algorithms for mission control, guidance, and vehicle control before integrating the software with the AUV.

D. MISSIONS

Autonomous vehicles will operate in environments where humans cannot or do not wish to go. The Defense Advanced Research Projects Agency (DARPA) has identified over 70 military missions especially suited for AUV execution (Bane and Ferguson, 1987), and this simulator incorporates a representative sampling of these tasks. The simulator's mission control software is divided into four main categories: charting, reconnaissance, surveillance, and covert payload delivery.

In each category, the mission controller executes the algorithms required to maneuver the AUV to the desired location, perform its required tasks, and return the vehicle to its starting position. Additional algorithms handle other tasks or emergencies such as path planning, uncharted shallow water, or close contacts.

Smaller "missions" test the NPS AUV guidance and control systems. These tasks, subsets of the larger missions, examine the vehicle's ability to transit and navigate in the test pool. This starts with simple maneuvers such as crossing the pool or circumnavigating it, and builds into more complicated sequences requiring the vehicle to execute depth changes, to pass through specific coordinates, and to maneuver for collision avoidance. Since the NPS AUV is capable of hovering, additional missions will be developed requiring the vehicle to switch between its propulsion and hovering systems for collision avoidance or reconnaissance.

E. CONTROL SYSTEM ARCHITECTURE AND LANGUAGES

The NPS AUV is controlled by a hierarchical system architecture that divides control among three areas: the mission level, the guidance level, and the execution level. Each section exchanges data or commands with its adjacent level and is responsible for monitoring or executing a specific portion of the AUV's mission.

1. The Mission Level

The mission level is the interface between the human user and the AUV. This level, written in the KEE software development shell, presents a menu of mission choices for the user's selection. After a choosing a mission, the user provides critical parameters for the AUV to execute. The mission level plans the execution of the task and issues the appropriate commands to the guidance level. Execution begins after initial path planning has been completed, and the mission level supervises the successful completion of the user's assigned tasks.

2. The Guidance Level

The simulator's guidance level receives orders from the mission level and generates guidance commands for the AUV's execution level. This software is written in Common LISP and is actually a set of modular procedures implemented according to the tasks selected by the mission level. One important procedure of this level is a global path planner using a best-first search algorithm to plot an AUV track which the vehicle follows to avoid charted obstacles. When the vehicle begins to transit toward its goal, the guidance level requires additional mission-level commands in order to invoke the correct procedures.

A second important procedure of the guidance level receives sensor inputs for processing and interpretation. Sensor data may require the guidance level to modify its commands to the execution level to avoid collisions or to take advantage of unanticipated mission opportunities.

3. The Execution Level

The execution level is written in the C language and is the lowest level of simulator control. This level receives guidance commands and executes routines to maneuver the AUV to the correct depth, course, and speed. The execution level updates the graphics displays (between ten and thirty times per minute) to reflect the mission's current status and passes the AUV's latest position up to the guidance and mission levels.

F. SIMULATION FACILITIES

The AUV simulator runs on a LISP machine and an IRIS graphics workstation. Two models of each machine were available for this thesis and were used to demonstrate performance variations and code compatibility.

1. Texas Instruments Explorer II LISP Machine

The TI Explorer II was chosen to execute the mission and guidance levels of simulator control. This machine is an advanced single-user workstation that uses the KEE software development shell to support the generation of large-scale and complex artificial intelligence programs. The programming environment includes very high speed proprietary processors, a large memory, sophisticated caching and memory-management systems, high-resolution black-and-white graphics, and networking facilities. The KEE shell gives the user a productive and intuitive programming environment for developing large and complex applications. (TI Explorer II User's Manual, 1988.)

2. Symbolics LISP Machine

The Symbolics 3675 LISP machine is also an artificial-intelligence workstation with the KEE software development shell. It has many of the same features as the TI LISP machine with additional support for image processing. Although the Symbolics LISP machine is slower than the TI, its image-processing capability will be incorporated into future research. (Symbolics User's Manual, 1987.)

3. IRIS 2400T Workstation

The Silicon Graphics IRIS-2400T graphics workstation is used for the original AUV simulator execution level and display. This system is a Unix-based high-resolution 1024 x 768 color display processor optimized for graphics applications. Its robust and highly efficient capabilities are primarily embedded in hardware instead of the more common software implementation. The system's fast execution is supported by an applications/graphics processor, a hardware matrix multiplier pipeline (the "Geometry Engine"), and a 32-bitplane raster subsystem. This workstation's specialized capabilities and speed make it particularly suitable for real-time displays. (IRIS User's Guide, 1986.)

4. IRIS 4D/70GT Workstation

The newer AUV simulator runs on the IRIS 4D/70GT graphics workstation. This machine, a third-generation descendant of the IRIS 2400T, is the result of extensive hardware and software design improvements. The new system pipeline architecture uses multiple RISC-based CPUs with a high-speed 64-bit data bus and a 96-bitplane raster subsystem. In addition to a much faster hardware Geometry Engine, the Unix-based software supports a style of object-oriented programming that greatly speeds image processing and updating. The system readily supports a higher update rate for a real-time AUV simulation while simultaneously incorporating graphics lighting and shading models. (IRIS 4D User's Guide, 1988.)

G. SUMMARY

This chapter discusses the problems of this thesis in detail and outlines proposed solutions. Several AUV models are described and differences in their mathematical and dynamic models are contrasted. The AUV simulator missions and environments are discussed and the system's architecture and programming languages are explained. Finally, the simulation's LISP machine and IRIS graphics workstation facilities are listed.

IV. AUV SIMULATOR DESCRIPTION

A. INTRODUCTION

This chapter describes the simulation software. The description starts with an explanation of the hierarchical AUV software architecture and how each level of the hierarchy carries out its tasks. The mission level of the software is discussed first, showing how the user inputs objectives to enable the controller to plan a mission and issue commands to the guidance level. The guidance section of this chapter explains how mission-level commands are implemented by the guidance level and how the results are represented at the execution level. A description of the software's execution level is followed by an explanation of the communications code that links all machines and software modules. The last section of this chapter is a user's manual; this manual repeats portions of the MacPherson (1988) manual for clarity and continuity.

B. SOFTWARE ARCHITECTURE

This thesis has preserved the hierarchical software architecture implemented by MacPherson (1988), as can be seen in Figure 4.1. The top level of the AUV software architecture is the mission level-- a knowledge base implemented using the KEE software development shell. The user interacts directly with this knowledge base by selecting a mission and then supplying additional information when

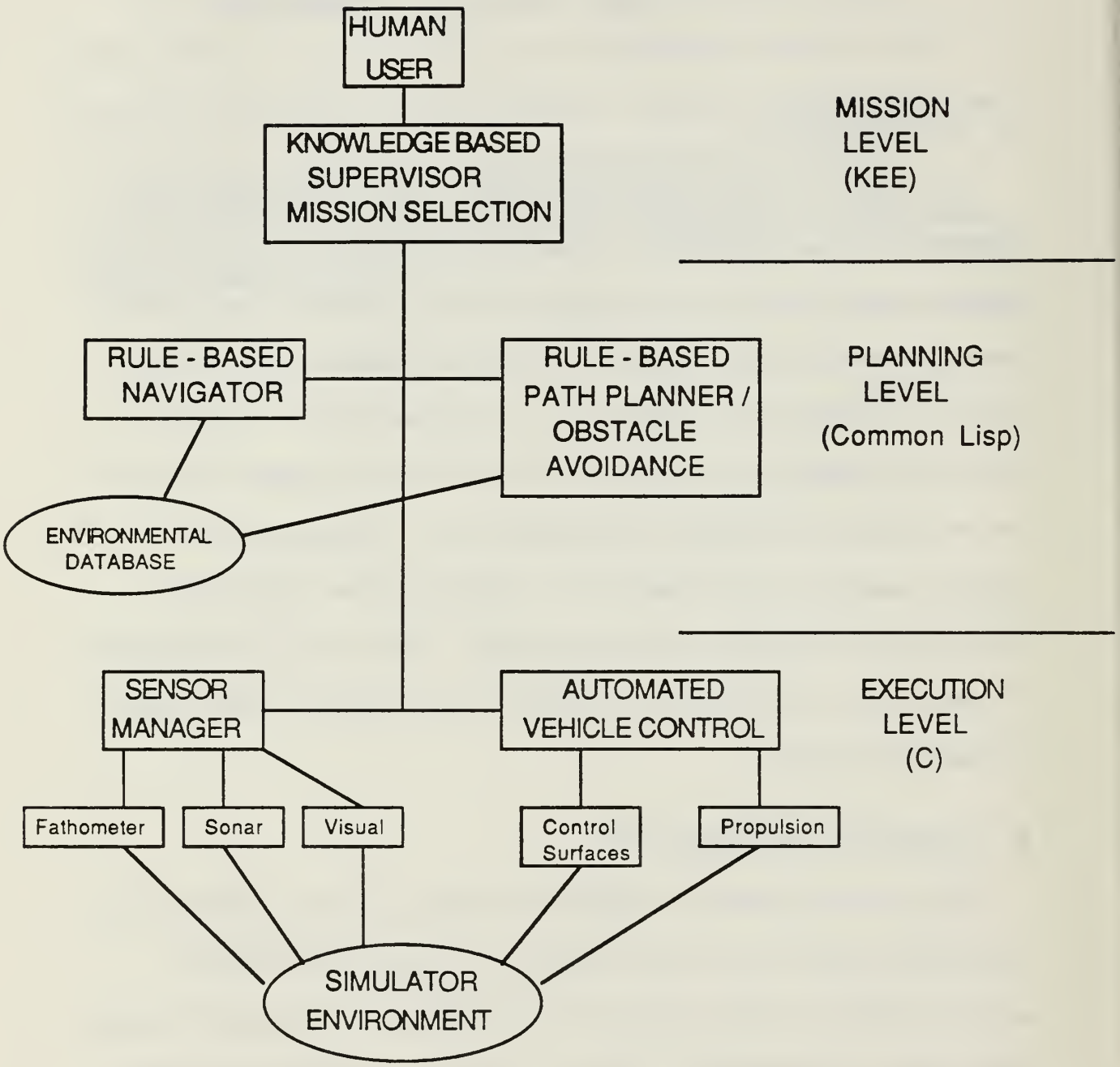

Figure 4.1 Hierarchical System Architecture

prompted. Once this information has been acquired, the simulator operates autonomously to carry out the user's mission and report its completion.

The simulator begins by accessing the second level in its hierarchy. The mission level starts its planning by passing its parameters down to the path planner and navigator. The planner and navigator are guidance-level Common LISP software modules which consult the guidance level's environmental database to select a path to the mission's goal and report this path back to the mission level.

Once supplied with the mission parameters and a path to the mission's goal, the guidance level communicates with the hierarchy's third level-- the execution code. Maneuvering parameters are interpreted by the execution-level autopilot as control surface commands that put the simulator's AUV on the path's course, speed, and depth. The execution level software includes sensor modules that provide simulated electronic, acoustic, and visual environmental inputs to the AUV. These inputs are passed back up the hierarchy to the navigator and the mission supervisor where the data is analyzed and acted on.

C. THE MISSION LEVEL

The KEE (Knowledge Engineering Environment) software development shell organizes the mission level. This powerful software development tool runs on the TI Explorer II or the Symbolics LISP machines and is used to place the simulator's many LISP functions into an easily-accessible structure. Although KEE includes a proprietary rule-based language, this simulator uses no rules to set up its relations. The structure of this knowledge base graphically links related missions on a tree diagram that can be traversed by the user with a mouse; the user makes a selection

and is prompted for additional information that is used to plan and execute the mission. Additional KEE utilities load the appropriate LISP files, display messages, update the knowledge base, and display the mission tree. The missions in this tree are created using a template system described in Chapter IV of MacPherson (1988).

The first NPS AUV missions will test the vehicle's propulsion and control surfaces. Once the mechanical systems operate satisfactorily, more sophisticated missions will be implemented to evaluate the AUV's ability to operate autonomously in a variety of situations. One of these missions under development is reconnaissance-- the AUV will be required to move about the pool, to locate and map obstacles for later analysis, and to trail moving objects while recording sensor information.

D. THE GUIDANCE LEVEL

The guidance level is written in Common LISP and runs on either the TI Explorer II or the Symbolics. These software modules, consisting of the mission navigator and path planner, receive orders from the mission level and provide guidance commands to the execution level. The first order from the mission level passes the mission's start and goal coordinates to the path planner. The path planner conducts a best-first search (Barr and Feigenbaum, 1981) to produce a series of coordinate subgoals which the vehicle will follow. These subgoals are passed to the simulation navigator and the mission's autonomous execution begins. Using frequent data exchanges via the communications interface, the navigator provides the execution level with information on the next subgoal, the autopilot course/speed/depth, and the command required to execute the current phase of the

mission. The communications interface passes back execution-level data on sonar contacts, the vehicle's position, and the depth of water under the AUV's keel. The guidance level processes this data and modifies the mission commands as necessary for the next data exchange.

The guidance level software is actually a number of LISP modules, each designed to perform a specific part of a mission. For example, the "transit" module contains code that commands the vehicle to move from the mission's starting coordinate to its next subgoal. This module executes from subgoal to subgoal until the AUV has reached its final goal.

E. THE EXECUTION LEVEL

The execution level is written in C and runs on the IRIS 2400T or the IRIS 4D/70GT graphics workstation. This level is the lowest level of AUV control; it executes either manual or autopilot commands to update vehicle and environmental displays. In autopilot mode, the execution level receives guidance-level commands for the location of the next mission subgoal, AUV course/speed/depth, and the mission phase. The execution level code interprets these commands, positions each control surface to achieve the AUV's parameters, and updates the graphics display to show the vehicle's current orientation.

At each update, the execution level passes sensor information up to the guidance level. This data is processed and can be used to alter the next set of guidance commands. An example of this occurs when the AUV's sensors report "uncharted" shallow water or obstacles (features unknown to the navigator's

environmental database) causing the guidance level to alter its commands, reposition the AUV, and prevent a collision.

1. The Original AUV Graphics Display

The original AUV display (MacPherson, 1988) is the primary means for a user to observe the performance of a vehicle as it executes an open-ocean mission. The display contains four sections: an "out the periscope" view of the environment, a navigation chart, a sonar screen, and a control panel. This presents the operator with a visual perspective from the AUV's point of reference, a plot showing the AUV's position, course, and goal, a sonar display of the contact situation, and a representation of the AUV's control surface positions. (See Figure 4.2.)

The upper left portion of the display is the periscope view. The AUV's periscope can be trained in azimuth or in elevation; it shows the vehicle's environment in either low- or high-power magnification. Several ships, islands, and buoys are included in this environment to provide obstacles for the AUV to avoid. When the AUV submerges, this periscope is automatically "lowered" and secured at a depth of fifty feet; the periscope is "raised" at fifty feet when the AUV returns to the surface.

The upper right portion shows the AUV's active sonar display. The execution-level code simulates an active sonar pulse transmission; contacts are shown on the display as black dots. Two cursors on the sonar display represent the vehicle's course (black) and the bearing on which the periscope is trained (green).

Figure 4.2 Original AUV Simulator Control Panel

The contact situation is updated every cycle and this information is passed up to the guidance control level.

The display's lower left portion depicts a navigation chart representing the simulator's environment and the vehicle's position. During autopilot control, a red "X" marks the AUV's position, the simulator's "start" and "goal" coordinates are outlined with red circles, and a red line shows the AUV's course. The chart uses shades of blue to show different water depths; land is displayed in black. As the guidance level executes its mission, the chart updates the AUV's position and displays the autopilot course, speed, and depth.

The display's center and lower right portions contain control panel information. Propulsion status is shown with a "battery charge" gauge to simulate the AUV's power supply; an alarm sounds when the vehicle reaches a low battery condition. Panels below the battery charge gauge show the AUV's actual course, speed, depth, pitch angle, and control surface positions so that the user can follow the vehicle's reactions to guidance-level commands.

2. The NPS AUV Graphics Display

The NPS AUV display has been adapted from a model programmed by Schwartz (1988) and is the user's means to observe the vehicle's performance during small-scale test missions. This simulation display consists of two sections: a representation of the AUV in a "test pool" from the perspective of an external viewer, and a control panel display showing the vehicle's and observer's parameters.

This graphics display is intended to simulate actual test conditions for the NPS AUV and is designed to show a realistic presentation of the vehicle in its

environment. The external perspective shows the NPS AUV (approximately five feet long) in a test pool (approximately 120 feet by 60 feet by eight feet deep) with several cylindrical obstacles. For path-planning purposes, the lower-left corner of the pool is the vehicle's coordinate origin with the pool's longer dimension on a north-south axis. The display allows the user to view the AUV's orientation and control surface positions as it maneuvers inside the test pool.

The control panel is similar to the original AUV control panel display. In addition to vehicle course, speed, depth, and control surface positions, it also gives the coordinates of the viewer's position with respect to the AUV. In manual control, the user can "shift position" to observe the vehicle from different perspectives as it executes its missions.

F. COMMUNICATIONS SOFTWARE

The execution-level code on each IRIS graphics workstation requires communications support for data exchanges with the guidance-level code on the LISP machine. Both communications modules link a graphics workstation with a LISP machine via an Ethernet cable; each module passes the same data types and structures in slightly different formats. The user selects the machines on which the simulation will be run; this determines which portions of the communications modules will be used to support the simulation.

The original communications code is adapted from MacPherson (1988); this thesis did not intend to alter that communications subsystem when the simulation was extended from one IRIS workstation to another. However, there is a significant difference between the hardware architecture and the operating systems of the two

IRIS machines-- and initial attempts to port the code were unsuccessful. Another thesis (Shannon and Teter, 1989) solved the problem of exchanging simulation data between a Symbolics machine and the IRIS 4D/70GT; that code was adapted for the NPS AUV simulator. Although additional effort will be able to convert the original communications code, its algorithms are not efficient and, as described below, this research used a different approach.

Regardless of the machine-specific algorithms, the information exchange between a LISP machine and an IRIS workstation allows the guidance level to send commands to control the execution level; the execution level uses the communications code to send sensor data back to the LISP machine for analysis. This information exchange executes in a loop that occurs about every three seconds. After carrying out its initial mission commands, the execution level passes to the guidance level a data package containing the AUV's present course, speed, and depth, the depth under the AUV's keel, and sonar contact bearing/range information. The LISP machine analyzes this data and sends back the mission phase command, the coordinates of the next subgoal, and the autopilot course, speed, and depth required to reach that subgoal.

This communications code is not critical to the success of the NPS AUV, so emphasis was placed on implementing a functional solution instead of a robust, efficient subsystem. While both versions of the communications code caused considerable problems during simulator implementation, these subsystems will not be required for actual AUV operations. The NPS AUV architecture design uses a

single processor; data is passed between the guidance and execution levels in a much quicker and more reliable structure.

1. The Original Communications Software

The software for this portion of the simulator is described in Barrow (1988) and provides standard routines to support communications between different computers connected via Ethernet and the Transmission Control Protocol/Internet Protocol (TCP/IP) systems (Comer, 1988).

The server portion of this package is written in C and runs on the IRIS 2400T workstation. The software supports full-duplex communications by sending data through one port and receiving data through a second port. These ports are linked to TCP/IP sockets and processes are spawned to service each socket. Data is exchanged through an IRIS shared-memory segment of one Mbyte to communicate between the IRIS execution level and the LISP-machine guidance level.

The client portion of the package is written in Common LISP and runs on either the TI Explorer II or Symbolics LISP machines. These routines manipulate an instantiation of the IP::TCP-HANDLER flavor of the system's TCP/IP software (Symbolics User's Manual, 1987; TI Explorer II User's Manual, 1988); two-port connections are established with a specific IRIS server to send and receive floating point numbers, integers, and characters.

When either the client or the server transmit data, the information is sent via a series of Ethernet packets; each packet contains a single four-byte data segment. Each Ethernet packet has a minimum 512-byte length so most of the packet is unused. The communications code execution is comparable to the IRIS

2400T display update rate so this inefficiency does not inhibit the simulation's execution, but this scheme can be improved to send all data in a single Ethernet packet.

2. The NPS AUV Communications Software

The original communications software required extensive modifications to run on the newer IRIS 4D/70GT workstation. After considerable experimentation, these modifications were judged to be beyond the scope of this thesis so another approach was investigated. Thesis work by Shannon and Teter (1989) involved similar communications code for a simulator application and the server portion of that code has been adapted to support the NPS AUV simulator.

The client code on the LISP machines is largely unchanged, but two additional algorithms have been added to read and write the different data format that is exchanged with the newer IRIS workstation. The read function parses the execution-level data package into its individual components and arranges these components in a list for analysis by the guidance-level routines. The write function places the guidance-level commands in a formatted message that is transmitted to the IRIS machine.

The server code on the IRIS machine supports semi-duplex communications-- two ports are still used on the LISP machines but all data is transferred through a single port on the IRIS workstation. Data exchange between the client and the server is always sequential so this does not affect the execution or guidance levels. Since all data is transferred in a single formatted message, this system makes more efficient use of the Ethernet packet size and the data exchange

proceeds at a much faster rate. (The LISP code for the client side of this system is shown in the Appendix.)

G. USER'S MANUAL

The NPS AUV simulator is menu-driven and prompts the user for input. This manual assumes the user has a basic familiarity with the Symbolics and TI Explorer II LISP machines as well as both IRIS graphics workstations. Some experience is required with the KEE expert system shell and the UNIX operating system in order to start up and secure the simulator, but not for its operation. Since this thesis works with a variety of machines and implementations, different portions of code will have to be loaded and executed depending on the combination of LISP machine and IRIS workstation the user desires. This manual also repeats portions of the MacPherson (1988) user's manual for continuity.

1. Graphics Workstation Operations

a. The Original AUV-- IRIS 2400T

The IRIS 2400T simulator can be operated in either the manual or autopilot mode. To start the simulation, "log on" to the IRIS 2400T side terminal and then transfer to the directory */work/nordman/thesis/symbolics/subsim* or */work/nordman/thesis/ti-explorer/subsim*. Start the simulator in manual by entering the command *sub* on the side terminal followed by a carriage return. After reading the initial display on the main terminal, press any mouse button to begin. The AUV simulator starts with the vehicle in manual control at a depth of *zero feet*, speed *zero knots*, and course *North*.

All manual control of the AUV is effected with the main terminal keyboard and its mouse buttons. To assist with this operating mode, a user's *Help display* may be toggled over the chart display by pressing the I-key. This display lists the various simulator controls that are explained below.

The AUV rudder is operated with the keyboard left-arrow and right-arrow keys. The sternplanes control the vehicle's pitch angle; the D-key places these planes on dive and the S-key places the sternplanes on rise. The bowplanes are used to make small changes to the AUV's depth without a pitch angle; these planes are manipulated with the B-key for dive and the C-key for rise. AUV speed is raised by the up-arrow key and lowered by the down-arrow key.

The AUV periscope has a number of features. The scope's magnification power is shifted by using the H-key for high power and the L-key for low power. The left mouse button rotates the periscope counterclockwise and the right mouse button rotates the scope clockwise. The middle button will raise the scope elevation angle while the simultaneous right and left mouse buttons will lower the elevation angle. As the AUV is surfaced and submerged, the periscope will automatically lower as the vehicle proceeds deeper than fifty feet; the periscope will be raised when the vehicle ascends shallower than fifty feet.

The autopilot is started by pressing the A-key on the main keyboard. The side terminal will indicate that the IRIS server is waiting to connect to either *expl* (the TI Explorer II) or *syml* (the Symbolics LISP machine); a message will also instruct the user to start the KEE portion of the simulator to connect the LISP client with the IRIS server.

The autopilot execution can be interrupted by pressing the Q key; this will terminate the autopilot but the simulator will return to its manual mode. The autopilot cannot be restarted (due to the system's shared memory constraints) but the simulator may be stopped by pushing all three mouse buttons simultaneously. Prior to restarting the IRIS-2400T simulator autopilot, ensure the previous socket connection has been correctly broken. To do this, list the current processes with the Unix command *ps -ax* and stop any */work/nordman...* communications daemons with the *kill* command. (Inexperienced users may require assistance for this step.)

b. The NPS AUV-- IRIS 4D/70GT

The NPS AUV simulator may also be operated in the manual or autopilot modes. To start the simulation, "log on" to the side terminal of the IRIS 4D/70GT and transfer to the directory */usr/work/nordman/thesis/symbolics/auvsim* or */usr/work/nordman/thesis/ti-explorer/auvsim*. Start the program in manual by entering the command *auv* on the side terminal followed by a carriage return.

The simulated AUV starts *on the surface* at a speed of *25 rpm* on course *East*. All manual control in this simulator uses the mouse to manipulate markers on the control panel at the right side of the main terminal display. To alter the viewer's perspective or to change AUV parameters, move the mouse arrow over the control panel marker for that parameter, press and hold the left mouse button, and drag the marker to the desired new value of that parameter. (Changes to the viewer's perspective should be executed slowly or the user may lose his own perspective in the display.) At very low speeds, the AUV may slowly roll from

port to starboard; raising speed will restore control surface effects on the mathematical AUV hydrodynamic drag model and should damp out this motion.

The autopilot is started by pressing the A-key on the main keyboard. The side terminal will indicate that the IRIS server is waiting to connect to either *exp1* (the TI Explorer II) or *sym1* (the Symbolics LISP machine); a message will also instruct the user to start the KEE portion of the simulator to connect the LISP client with the IRIS server.

The autopilot execution can be interrupted by pressing the Q-key; the autopilot cannot be restarted at this point but manual control of the AUV is available. Prior to restarting the IRIS 4D/70GT simulator autopilot, ensure the previous socket connection has been correctly broken. To do this, list the current processes with the Unix command *ps -al* and stop any */usr/work/nordman...* send/receive communications daemons with the *kill* command. (Inexperienced users may require assistance for this step.)

2. LISP Machine Operations

a. TI Explorer II

The TI Explorer II must be loaded with the KEE expert system software shell. (It should be available by entering SYSTEM-K.) If the shell is not loaded, a cold boot of the machine will be required; inexperienced users should get help at this point. When KEE is available, "log on" in the LISP Listener and then press the SYSTEM-K combination to move to the KEE desktop. (See Figure 4.3.)

Figure 4.3 KEE Desktop with a Proposed Reconnaissance Mission

Use the mouse to point at the KEE latchkey icon at the screen's upper left corner and push the left mouse button.

A pop-up menu will appear offering *KEE Commands*; select the *Load KB* command by pointing at the command with the mouse and pushing the left mouse button. A KEE "typescript window" will appear requesting the name of the knowledge base to be loaded; enter *exp3:nordman;auv.u* followed by a carriage return. KEE will load the AUV knowledge base and then will load the LISP code files containing the guidance level functions. (The loading process will take about ten seconds.) Once the files are loaded, use the mouse to point to the word *AUV* in the knowledge base window and press the left mouse button. Another pop-up menu will offer *KB Commands*; use the left mouse button to select the *Display* option. The AUV Simulator Mission Tree will be drawn in a window labelled *The Graph of the AUV Knowledge Base.*

Select an AUV mission by using the mouse to point to the desired generic mission at one of the leaves of the Mission Selection Tree. Press the left mouse button to choose the mission; a pop-up menu of *Unit Commands* will appear. Point at the *Send Message* option with the mouse and push the left mouse button again; a pop-up menu of *Message Types* will appear. Point the mouse to the message that starts the desired mission and press the left mouse button a final time.

The AUV simulator mission-level code will ask the user several questions about mission parameters. These questions will appear in the KEE typescript window and should be answered by entering answers to each of these questions followed by a carriage return. When the KEE typescript window

announces that it has connected with the appropriate IRIS workstation, move to that workstation and press a carriage return on the side terminal. The IRIS side terminal will notify the user that it has sent initial parameters to the LISP machine and the mission-level code will begin a best-first search to determine the AUV's path. When the KEE typescript window displays the message *autopilot course on first leg is:*, press another carriage return on the IRIS side terminal to begin autonomous simulator execution.

If necessary, the AUV simulator can be stopped by entering CONTROL-ABORT. This will complicate the orderly shutdown of the TI Explorer II communications sockets and should be avoided. Once the simulator has completed its mission, return to the LISP Listener and enter the command *(end-con)* to break the TI-IRIS socket connection. The simulation may be restarted by re-entering the appropriate commands to start another mission from the KEE Mission Tree.

b. Symbolics

On the Symbolics LISP machine, ensure the KEE expert system shell software is loaded. (It is accessed by entering SELECT-K.) If the shell is not loaded, a cold boot of the machine will be required; an inexperienced user should get staff assistance at this point. Once KEE is available, "log on" in the LISP Listener and then press the SELECT-K key combination to move to the KEE desktop. (See Figure 4.3 on page 49.) Use the mouse to point at the KEE latchkey icon at the screen's upper left corner and push the left mouse button. A pop-up menu will appear offering *KEE Commands*; select the *Load KB* command

by pointing at the command with the mouse and pushing the left mouse button. A KEE "typescript window" will appear requesting the name of the knowledge base to be loaded; enter *sym4:>nordman>auv.u* followed by a carriage return. KEE will load the AUV knowledge base and then will load the LISP code files containing the guidance-level functions. (This process will take approximately thirty seconds.) Once the files have completed loading, use the mouse to point to the word *AUV* in the knowledge base window and press the left mouse button. Another pop-up menu will offer *KB Commands*; use the left mouse button to select the *Display* option. The AUV Simulator Mission Tree will be drawn in a window labelled *The Graph of the AUV Knowledge Base.*

Prior to selecting a mission, an IRIS terminal must be chosen for the simulation. Ensure that either the IRIS 2400T or the IRIS 4D/70GT has been placed in its autopilot mode and is waiting to connect to the Symbolics LISP machine (*sym1*). On the Symbolics machine, return to the LISP Listener window (SELECT-L, not the KEE LISP Listener) and select the appropriate IRIS workstation by entering the commands *(choose-iris 'iris2)* (for the IRIS 2400T) or *(choose-iris 'iris5)* (for the IRIS 4D/70GT). This will start the TCP/IP software on the LISP machine and locate the correct IRIS port(s) for data exchange. The user should then return to the KEE environment by entering SELECT-K.

Back in the KEE shell, select a mission by using the mouse to point to the desired generic mission at one of the leaves of the Mission Selection Tree. Press the left mouse button to choose the mission; a pop-up menu of *Unit Commands* will appear. Point at the *Send Message* option with the mouse and push

the left mouse button again; a pop-up menu of *Message Types* will appear. Point the mouse to the message that starts the desired mission and press the left mouse button a final time.

The AUV simulator mission-level code will ask the user several questions about mission parameters. These questions will appear in the KEE typescript window and should be answered in that window by first selecting the window with the mouse and then entering answers to each of these questions followed by a carriage return. When the KEE typescript window announces that it has connected with the appropriate IRIS workstation, move to that workstation and press a carriage return on the side terminal. The IRIS side terminal will notify the user that it has sent initial parameters to the LISP machine and the mission-level code on the LISP machine will begin a best-first search to determine the mission's path. When the KEE typescript window displays the message *autopilot course on the first leg is:*, press another carriage return on the IRIS side terminal to begin autonomous simulator execution.

If necessary, the AUV simulator can be stopped by entering CONTROL-ABORT. This will complicate the orderly shutdown of the Symbolics communications sockets and should be avoided. Once the simulator has completed its mission, return to the LISP Listener and enter the command *(end-con)* to break the Symbolics-IRIS socket connection. The simulation may be restarted by securing and restarting an IRIS simulator, re-entering the appropriate LISP machine command to choose an IRIS workstation, and then starting another mission from the KEE Mission Tree.

H. SUMMARY

This chapter describes the simulator's operation. The first section starts with an overview of mission execution and then stresses the specific workings of the software architecture hierarchy. Subsequent sections provide detailed discussions of the mission, guidance, and execution levels of this hierarchy for two types of IRIS graphics workstations. One section describes both types of communications software used in this thesis, explains the reasons for their use, and discusses their differences. The final section of this chapter contains the User's Manual to assist in operating different combinations of the four machines.

V. EXPERIMENTAL RESULTS

A. INTRODUCTION

This chapter evaluates the experimental results of the AUV software conversion and examines sample runs on the simulator's various combinations of machines.

B. AUV SIMULATION FACILITIES

1. The Original Simulator

The original AUV graphics simulation (on the IRIS 2400T) now runs under either the Symbolics or the TI Explorer II LISP machines. The TI Explorer II is the faster machine (by roughly 50%) but manufacturer's updates to its operating system have historically caused problems with the simulator's communications code port requests. The Symbolics machine is a reliable backup to the TI Explorer II.

2. The NPS AUV Simulator

The NPS AUV simulator (on the IRIS 4D/70GT) now runs under autopilot control from either the Symbolics or TI Explorer II LISP machines. The guidance-level LISP code still performs as described in MacPherson (1988) but the execution-level output is now displayed from an external perspective and in a more realistic simulation.

Several revisions to the execution-level code improve the simulator's realism. The NPS AUV program was optimized to run on the faster IRIS 4D/GT70 workstation and its algorithms take advantage of the machine's processing capability and greater speed. This is clearly demonstrated in the simulation's graphics, where lighting and shading models give the AUV and its environment a depth perspective that is not evident in the "flat" graphics display of the IRIS 2400T. While the earlier simulator rotates the environmental scene about the viewer, this simulator allows the viewer to "move about" the display and to examine it from a nearly unlimited variety of angles, distances, and elevations. Although this type of display requires much more computer processing, the IRIS 4D/70GT manipulates its perspectives faster than the IRIS 2400T can run its simpler simulation.

The user interface for the NPS AUV simulator is easier to understand and simpler to operate. The IRIS 2400T program accepts input through twenty different keys and mouse buttons (and uses a help menu to assist the user), but the IRIS 4D/GT70 display is completely mouse-driven with one button. While the older vehicle requires continuous user input to reach its desired parameters, the newer simulation immediately accepts and displays the user's commands and then applies them to the hydrodynamic equations. The user is free to examine the vehicle's response without having to provide additional input.

The NPS AUV program allows the user a greater flexibility in starting and observing a mission; it also permits more missions to be run in the same amount of time. While the older simulation starts each mission from the same location, the NPS AUV simulator allows the user to start from any depth or position in the test

pool. Instead of observing the older AUV missions on a two-dimensional plot, the newer simulator allows the user to choose an initial viewing position before beginning the mission. This useful feature permits observation from a number of perspectives to examine control surfaces, thruster operation, or restricted maneuvering. The user is able to conveniently and quickly examine the AUV's simulated operation in exactly the same way as the actual vehicle will be observed. This not only exceeds the capabilities of the older simulator but also can produce reduced testing requirements for the actual NPS AUV.

The NPS AUV simulator autopilot uses the same methods as the older simulator to initialize its missions. After selecting the viewer perspective and the vehicle's initial position, the user inputs the same mission parameters as required by the original simulation. Although the NPS AUV is operating in a smaller environment, the vehicle still has a significant number of obstacles to avoid when conducting path-planning. The best-first path-planning algorithm operates satisfactorily (and identically) in either environment to guide the vehicles around obstacles and reach a goal. Its test-pool transit is shorter, but the NPS AUV can encounter more obstacles than the older vehicle, so the path-planner generates more complicated routes with a number of abrupt maneuvers to reach the vehicle's goal. Once the mission's execution is started, the vehicle maneuvers to reach its goal while the viewer watches from a chosen perspective. The side terminals of both graphics workstations also display information on vehicle parameters and environmental conditions.

The NPS AUV simulator has been explicitly designed to test a variety of AUV models in a selection of environments. Significant software design effort was devoted to implementing a modular code structure that will allow the rapid addition of new features. Proficient programmers can alter the location of obstacles, add moving contacts, change the AUV's maneuvering characteristics, and test different types of controllers all by adding or replacing modules of simulator code.

The NPS AUV simulator presently runs much simpler missions than the earlier simulation. Although the test pool is filled with obstacles, there are no maneuvering contacts for the NPS AUV to track and avoid. The transit and reconnaissance missions are subsets of the original program's more complicated open-ocean missions, but these missions are being expanded to incorporate contact avoidance, dynamic path-replanning, switching between hovering and propulsion, and using sonar for environmental mapping. As different system designs and algorithms are developed for successive NPS AUV models, their performance can examined by adding those features into the test pool missions.

3. Communication Between IRIS and LISP Machines

The inter-computer communications code developed by Barrow (1988) is designed to run under the AT&T Unix System V implementation and was originally developed for the IRIS 2400T and IRIS 4D workstations. (The IRIS 4D is an earlier version of the IRIS 4D/70GT.) After several lengthy and unsuccessful attempts to port the Barrow code to the IRIS 4D/70GT, the Shannon and Teter (1989) communications code was substituted for the Barrow client/server system.

Nearly all of the Unix System V routines on the IRIS 4D/70GT appear to be compatible with earlier implementations; most of the code conversion only required updating library subroutine definitions. However, the IRIX operating system (IRIS 4D User's Guide, 1988) handles shared-memory operations in a different manner than the earlier IRIS 4D systems and it is unclear whether the Barrow shared-memory routines can be adapted to the IRIS 4D/70GT. Time constraints halted work on this code conversion; the Shannon and Teter routines were substituted.

A second (and more serious) set of problems arose after the operating system on the TI Explorer II was updated. The client communications code on either LISP machine obtains a socket port number before connecting with an IRIS workstation, but the newer version (4.1) of the Explorer II's operating system does not execute that port-number request-- the code would not even compile or load. A lengthy and complicated debugging procedure was unable to isolate and correct the problem, and the code was even re-written to use different communications protocols. Due to time constraints, this problem was finally avoided by porting the TI Explorer II communications code to the Symbolics LISP machine. Copies of the code and its error messages were sent to Texas Instruments for technical assistance, where it was determined the failure was due to an operating system deficiency (TI, 1989). An updated Explorer II operating system (version 4.2) has been obtained and the communications code will be adapted to this new implementation.

C. NPS AUV SIMULATOR OPERATION

The guidance-level routines developed by MacPherson (1988) are used to build the missions executed by the NPS AUV. These modular routines allow the user to design more complex AUV missions by simply adding specific tasks to the mission-level code. An evolving example of this is the NPS AUV reconnaissance mission.

The first version of the reconnaissance mission only required the AUV to perform a surface transit between two points. After satisfactory performance was demonstrated, this was modified to require a submerged transit along a specified path with the vehicle returning to its starting point upon completion. The current reconnaissance mission requires the AUV to perform a submerged transit to its goal, surface to simulate a photographic and electronic sweep of the environment, submerge to return to its starting point, and surface for pickup.

The NPS AUV simulator shows that the vehicle can accomplish a simple test-pool reconnaissance mission with the same performance as the original vehicle in its open-ocean reconnaissance, and the simulator's missions can easily be modified to execute more complex requirements.

D. SUMMARY

This chapter presents a summary of the NPS AUV simulator and its facilities. The capabilities of the two LISP machines are contrasted, the simulator's communications code is discussed, and the performance of the AUV mission-control routines is evaluated when the simulation uses the more powerful IRIS 4D/70GT graphics workstation.

VI. SUMMARY AND CONCLUSIONS

A. RESEARCH CONTRIBUTIONS

The NPS AUV simulator is an important tool for incorporating new autonomous control concepts and algorithms into the latest version of the NPS AUV. The KEE expert system shell provides an inexperienced user with an intuitive menu-driven system that allows rapid mission planning and execution. The shell also provides a powerful environment where programmers can modify the simulator's mission-level code and develop additional missions. The faster and more powerful IRIS 4D/70GT graphics workstation effectively simulates the AUV's actual operation with a real-time display of the vehicle's actions, and this workstation has the capacity to accommodate more complex AUV models or controllers. The vehicle simulations have been improved and modified to run on two different LISP machines or graphics workstations.

The simulator is a valuable test and debugging environment that will save countless hours of experimentation; it will also verify code reliability before the software is installed in the actual NPS AUV computer system. The new NPS AUV simulator provides the user with a wide choice of starting locations and viewing positions to thoroughly examine vehicle performance from many different perspectives. This viewing flexibility greatly reduces the risks, simplifies the logistics, and minimizes the costs of testing the NPS AUV in its ocean environment.

Although it was not intended to be a goal of the simulator research, a better communications system is available (Shannon and Teter, 1989) and has been adapted to this simulator. The system is faster and more efficient but it is invoked at a lower level of abstraction; it will be more difficult to modify when the simulator is updated to pass additional sensor data to the guidance level of control. While this simulator was developed as a research project, it is designed to be used as a tool. This means that a simple, easily-modified communications system is potentially more useful than a faster system with a more complex interface. The communications support of distributed simulation systems is vitally important for the rapid design and modification of these simulators; a more thorough discussion of this issue is presented in Barrow (1988).

B. RESEARCH EXTENSIONS

Several research extensions are discussed in MacPherson (1988) and should still be applied to this simulator. This includes a faster and more sophisticated path-planning algorithm, an AUV vision system for mapping and for contact classification, algorithms for inertial or terrain-following navigation systems, and an environment for examining the performance of high-resolution sonar systems.

This simulator must evolve along with the NPS AUV. The NPS AUV design team is developing more sophisticated AUV models and controllers as well as the hydrodynamic data to describe the performance of these vehicles. The simulator must incorporate the results of that research to present an accurate display of the latest AUV's hydrodynamic and maneuvering characteristics. A highly realistic simulation will produce valuable and timely feedback by quickly demonstrating the

potential problems or side effects generated by the design team's efforts. Specific examples of these modifications are the hydrodynamic data from the NPS Model 2 AUV, a "sliding-mode controller" for the NPS AUV's maneuvering system, and guidance-level software to control the vehicle's transition between propulsion and hovering modes.

An effective AUV must possess adaptability and the ability to replan its mission as a result of unexpected events. All missions in this simulator are still relatively simple and complex missions will require a more complex planner. For example, if the vehicle detects an obstacle or an "interesting" contact while executing an unrelated mission, it may have to interrupt its primary mission, switch to its hovering control mode, and investigate that object. The code required to implement the "interrupt" and "hover" decisions will require a rule-based expert system; this system can be constructed by supplementing the LISP routines with Prolog rules or with the rule-based features of an expert systems shell.

This simulator's modular design allows additional vehicle features to be quickly incorporated into the graphics display. This will provide a realistic demonstration of the performance of new AUV models and controllers and it will allow several different models to be compared in a laboratory environment. As the sophistication and variety of these models increases, the simulator's organization must also be updated to maintain a simple and user-friendly interface. This can be done by using menu-driven options to allow the user to select vehicle models and control characteristics before starting the simulation.

APPENDIX

This appendix lists the client-side LISP communications code of the NPS AUV.

```
;;; -*- Mode: LISP; Syntax: Common-lisp; Package: USER -*-

;;;    This is the program SYM-IRIS-COMM.LISP.  It provides the necessary
;;;    software on the Symbolics LISP machine to communicate with the IRIS
;;;    2400T or the IRIS 4D/70GT.  It is loaded in the KEE shell with the
;;;    knowledge base AUV.U.

;
;  "Talk" is an object to send and to receive data across a network.
;
;  usage : (send talk :init-destination-host 'iris2)  ; get remote host object
;          (send talk :start-iris)                    ; make connection
;          (send talk :put-iris data)                  ; send data
;          (send talk :get-iris)                       ; get data from remote host
;          (send talk :stop-iris)                      ; close communication
;          (send talk :reuse-iris)                     ; open closed communication
;          (send talk :change-iris-ports)              ; switch from IRIS2 full-duplex
;                                                        ; comms to IRIS5 semi-duplex


(defvar talk)


;
; library functions to be used by flavor conversation-with-iris.
;


(defmacro loopfor (var init test exp1)
  `(prog ()
      (setq ,var ,init)
      tag
         ,exp1
         (setq ,var (1+ ,var))
         (if (= ,var ,test) (return t) (go tag))))
```

```
(defun convert-number-to-string (n)
  (princ-to-string n))

(defun convert-string-to-integer (str &optional (radix 10))
  (do ((j 0 (+ j 1))
       (n 0 (+ (* n radix) (digit-char-p (char str j) radix))))
      ((= j (length str)) n)))

(defun find-period-index (str)
  (catch 'exit
    (dotimes (x (length str) nil)
      (if (equal (char str x) (char "." 0))
        (throw 'exit x)))))

(defun get-leftside-of-real (str &optional (radix 10))
  (do  ((j 0 (1+ j))
      (n 0 (+ (* n radix) (digit-char-p (char str j) radix))))
       ((or (null (digit-char-p (char str j) radix)) (= j (length str)))  n)))

(defun get-rightside-of-real (str &optional (radix 10))
  (do ((index (1+ (find-period-index str)) (1+ index))
       (factor 0.10 (* factor 0.10))
       (n 0.0 (+ n (* factor (digit-char-p (char str index) radix)))))
      ((= index (length str)) n )))

(defun convert-string-to-real (str &optional (radix 10))
  (+ (float (get-leftside-of-real str radix)) (get-rightside-of-real str radix)))


;
; Port number definitions:  IRIS2 uses full duplex comms so ports are set up for
;       this default.  IRIS5 uses semiduplex comms (the same port for send and
;       receive) and will have both ports set to *remote-port1*.
;



(defvar *remote-port1* 1027)                    ; this is the remote send port
(defvar *remote-port2* 1026)                    ; this is the remote receive port
(defvar *local-talk-port* 1500)                 ; this is the local send port
(defvar *local-listen-port* 1501)               ; this is the local receive port

;
;  Conversation-with-iris flavor definition.
;
:
```

```
; This definition is not restricted to IRIS, but it can be
; used with any host as long as the remote host does not
; already use ports 1027 or 1026 for its own purposes.
;


(defflavor conversation-with-iris ((talking-port-number       *remote-port1*)
                                   (listening-port-number     *remote-port2*)
                                  (local-talk-port-number    *local-talk-port*)
                                  (local-listen-port-number *local-listen-port*)
                                   (talking-stream)
                                   (listening-stream)
                                  (destination-host-object)
                                         )
                                      ()
                                   :initable-instance-variables)

(defmethod (:init-destination-host conversation-with-iris)
         (name-of-host)
  (setf destination-host-object (net:parse-host name-of-host)))


(defmethod (:change-iris-ports conversation-with-iris)
         ()
  (setf talking-port-number     *remote-port1*)
  (setf listening-port-number   *remote-port1*))


(defmethod (:start-iris conversation-with-iris)
         ()
  (setf talking-stream
      (tcp:open-tcp-stream destination-host-object
                           talking-port-number
                           local-talk-port-number))
  (setf listening-stream
      (tcp:open-tcp-stream destination-host-object
                           listening-port-number
                           local-listen-port-number))
  (terpri)
  (princ "A conversation with the IRIS machine has been established.")
  (terpri))

(defmethod (:reuse-iris conversation-with-iris)
         ()
  (send self :start-iris))
```

```
(defun read-string (stream num-chars)
  (let ((out-string ""))
    (dotimes (i num-chars)
      (setf out-string (string-append out-string (read-char stream))))
    out-string))


(defmethod (:get-iris conversation-with-iris)
         ()
  (let* ((typebuffer    " ")
        (lengthbuffer "     ")
          (buffer         " ")
        (buffer-length 1))
    (progn
      (setf typebuffer
          (read-string listening-stream 1))
      (setf lengthbuffer
          (read-string listening-stream 4))
      (setf buffer-length
          (convert-string-to-integer lengthbuffer))
      (setf buffer
          (read-string listening-stream buffer-length))


     (cond ((equal typebuffer "I") (convert-string-to-integer buffer))
           ((equal typebuffer "R") (convert-string-to-real     buffer))
           ((equal typebuffer "C") buffer)
           (t nil)))))



(defvar *step-var* 0)



(defun my-write-string(string stream)
  (let* ((num-chars (length string)))
    (dotimes (i num-chars)
      (write-char (aref string i) stream))))
(defmethod (:put-iris conversation-with-iris)
         (object)
```

```
(let* ((buffer (cond
          ((equal (type-of object) 'bignum) (convert-number-to-string object))
            ((equal (type-of object) 'fixnum) (convert-number-to-string object))
          ((equal (type-of object) 'single-float) (convert-number-to-string object))
          ((equal (type-of object) 'string) object)
          (t "error")))

     (buffer-length  (length buffer))

     (typebuffer     (cond ((equal (type-of object) 'bignum) "I")
                       ((equal (type-of object) 'fixnum) "I")
                       ((equal (type-of object) 'single-float) "R")
                       ((equal (type-of object) 'string) "C")
                       (t "C")))

     (lengthbuffer   (convert-number-to-string buffer-length)))


   (progn
     (my-write-string typebuffer talking-stream)
     (send talking-stream :force-output)


     (if (= (length lengthbuffer) 4)
         (write-string lengthbuffer talking-stream)
         (progn
            (loopfor *step-var* (length lengthbuffer) 4
                 (write-string "0" talking-stream))

            (my-write-string lengthbuffer talking-stream)
              ))

     (send talking-stream :force-output)


     (my-write-string buffer talking-stream)
     (send talking-stream :force-output)

     )))

(defmethod (:check-iris conversation-with-iris) (size-io)
   (let* ((typebuffer)
          )
      (progn
         (setf typebuffer
            (read-string listening-stream size-io)))))
```

```
(defmethod (:stop-iris conversation-with-iris)
        ()
  (progn (send listening-stream :close)
       (send talking-stream :close))
  (terpri)
  (princ "A conversation with the IRIS machine has been closed.")
  (terpri))


(setf talk (make-instance 'conversation-with-iris))



(defun choose-iris (host-name)
  (cond
    ((equal host-name 'iris2)
     (send talk :init-destination-host host-name)      ;use iris2 as default output.
     (terpri)
     (princ "IRIS2 selected.")
     (terpri))
    ((equal host-name 'iris5)
     (send talk :change-iris-ports)                    ;select semi-duplex comm ports.
     (send talk :init-destination-host host-name)
     (terpri)
     (princ "IRIS5 selected.")
     (terpri))))


(defun start-con()
  (send talk :start-iris))

(defun get_data()
  (send talk :get-iris))

(defun send_float(single-float)
  (send talk :put-iris single-float))

(defun send_string(string)
  (send talk :put-iris string))

(defun end-con()
  (send talk :stop-iris))

(defun restart()
  (send talk :reuse-iris))
```

LIST OF REFERENCES

ART, 1989.

Applied Remote Technology *XP-21* advertising literature, 9950 Scripps Lake Drive, San Diego, CA, 1989.

ART Reference Manual, 1986.

ART Reference Manual, version 2.0, pp. 1-1 to 1-5, Inference Corporation, Los Angeles, CA, 1986.

Baker, 1989.

Baker, A. D. III, "Combat Fleets", U.S. Naval Institute *Proceedings*, v. 115, No. 5, p. 158, May 1989.

Bane and Ferguson, 1987.

Bane, G. and Ferguson, J., "The Evolutionary Development of the Military Autonomous Vehicle, *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, v. 5, pp. 23-27, June 1987.

Barr and Feigenbaum, 1981.

Barr, A. and Feigenbaum, E., *The Handbook of Artificial Intelligence*, pp. 58-64, William Kaufmann, Inc., Los Altos, CA, 1981.

Barrow, 1988.

Barrow, T., *Distributed Computer Communications in Support of Real-Time Visual Simulations*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1988.

Blidberg and Chappell, 1986.

Blidberg, D. R., and Chappell, S. G., "Guidance and Control Architecture for the EAVE Vehicle," *IEEE Journal of Oceanic Engineering*, v. OE-11, No. 4, pp. 449-61, October 1986.

Brady et al., 1984.

Brady, M., Gerhardt, L., and Davidson, H., "Artificial Intelligence and Robotics," *Robotics and Artificial Intelligence, Series F: Computer and System Sciences*, Springer-Verlag, v. 11, p. 47, 1984.

Comer, 1988.

Comer, D., *Internetworking with TCP/IP: Principles, Protocols, and Architecture*, Prentice-Hall, Inc., 1988.

Doty and Wachter, 1986.

Doty, D. C., and Wachter, R. F., "On a Blackboard Architecture for an Object-oriented Production System," *Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics*, v. 2, pp. 1626-30, 1986.

ECA, 1989.

Societe ECA *Pope* and *Epaulard* advertising literature, Toulours, France, 1989.

Eisenstadt, 1987.

Eisenstadt, S., "Navy Envisions $5 Billion ASW Minisub Fleet", *Defense News*, v. 2 No. 51, p. 1, 24 December 1987.

IRIS 4D User's Guide, 1988.

IRIS 4D User's Guide, Silicon Graphics, Inc., Mountain View, CA, 1988.

IRIS User's Guide, 1986.

IRIS User's Guide, Silicon Graphics, Inc., Mountain View, CA, 1986.

Jalbert, 1987.

Jalbert, J., "Low-Level Architecture for the New EAVE Vehicle," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, v. 5, pp. 238-243, June 1987.

KEE User's Manual, 1986.

KEE Software Development System User's Manual, version 3.0, pp. 1-1 to 1-4, Intellicorp, Mountain View, CA, 1986.

MacDonald, 1989.

MacDonald, G. L., *Model-based Compensator Design and Experimental Verification of Control Systems for a Model AUV*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1989.

MacPherson, 1988.

MacPherson, D. L., *A Computer Simulation Study of Rule-based Control of an Autonomous Underwater Vehicle*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1988.

Robinson, 1986.

Robinson, R. C., "National Defense Applications of Autonomous Underwater Vehicles," *IEEE Journal of Oceanic Engineering*, v. OE-11, No. 4, pp. 462-7, October 1986.

ROV '89, 1989.

ROV '89 Convention, San Diego, CA, 14-16 March 1989.

ROV News, 1989.

"DARPA Awards $23.9 Million Contract to Draper Labs," *ROV News*, v. 2, No. 1, p. 2, January 1989.

Schwartz, 1989.

Schwartz, M., *System Identification and Control for an AUV*, Master's Thesis, Naval Postgraduate School, Monterey, CA, March 1989.

Schweizer and Oravec, 1986.

Westinghouse Research and Development Center R&D paper 86-1C5-AVERS-P1, *Intelligent Control and Signal Processing for an Autonomous Underseas Vehicle*, Schweizer, P. F. and Oravec, J. J., 19 August 1986.

Sea Technology, 1989.

Mullen, C., "*Gemini 6000* ROV", *Sea Technology*, v. 3, No. 2, p. 15, February 1989.

Shannon and Teter, 1989.

Shannon, L., and Teter, W., *APS: An Autonomous Platform Simulator*, Master's Thesis, Naval Postgraduate School, Monterey, CA, June 1989.

Shevenell, 1987.

Shevenell, M., "Hardware & Software Architectures for Realizing a Knowledge Based System on EAVE," *Proceedings of the Fifth International Symposium on Unmanned Untethered Submersible Technology*, v. 5, pp. 220-8, June 1987.

Steele, 1988.

Steele, S., "Remotely Piloted Vehicles", U.S. Naval Institute *Proceedings*, v. 114, No. 6, p. 78, 27 June 1988.

SUBNOTES, 1988.

"*XP-21* AUV Test Platform" and "Underwater Resources *Sea Ferret* ROV," *SUBNOTES*, v. 4, No. 5, p. 2, September/October 1988.

SUBNOTES, 1989a.

"*Gemini 6000* Rov System Recovers Cockpit Recorder," *SUBNOTES*, v. 5, No. 2, p. 16, March/April 1989.

SUBNOTES, 1989b.

"ROV Conducts Oceanographic Research Off California's Coast," *SUBNOTES*, v. 5, No. 2, p. 19, March/April 1989.

SUTEC, 1989.

Scandinavian Underwater Technology *Sea Owl* and *Sea Twin* advertising literature, Stockholm, Sweden, 1989.

Symbolics User's Manual, 1987.

Symbolics 3600 Series User's Manual, Symbolics Inc., Concord, MA, 1987.

Tamaki, 1989.

Tamaki, U., "Free Swimming *PTEROA* For Deep Sea Survey," paper presented at ROV '89, San Diego, CA, 14-16 March 1989.

TI, 1989.

Telephone conversations between Texas Instruments Explorer II Technical Assistance and the author, 27-29 April 1989.

TI Explorer II User's Manual, 1988.

Texas Instrument Explorer II User's Manual, Texas Instruments Inc., Austin, TX, 1988.

Underwater Resources, 1989.

Underwater Resources *Sea Ferret* ROV advertising literature, Pier 26, Embarcadero, San Francisco, CA, 1989.

WHOI, 1986.

Woods Hole Oceanographic Institution Deep Submergence Lab *Argo* Development Program Final Report, June 1986.

Yoerger et al., 1986.

Yoerger, D., Newman, J., and Slotine, J., "Supervisory Control System for the *JASON* ROV," *IEEE Journal of Oceanographic Engineering*, v. OE-11, No. 3, pp. 392-400, July 1986.

INITIAL DISTRIBUTION LIST

		Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chief of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, DC 20350-2000	1
4.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5000	2
5.	Curricular Officer, Code 33 Weapons Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
6.	Professor Robert B. McGhee, Code 52Mz Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	9
7.	Professor Sehung Kwak, Code 52Kw Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1
8.	Professor Yuh-Jeng Lee, Code 52Le Computer Science Department Naval Postgraduate School Monterey, CA 93943-5000	1

9. Department Chairman, Code 69Hy — 1
Mechanical Engineering Department
Naval Postgraduate School
Monterey, CA 93943-5004

10. Professor R. Cristi, Code 62Cx — 1
Electrical and Computer Engineering Department
Naval Postgraduate School
Monterey, CA 93943-5004

11. United States Military Academy — 1
Department of Geography & Computer Science
ATTN: CPT Mark Fichten
West Point, NY 10996-1695

12. Naval Ocean System Center — 1
Ocean Engineering Division (Code 94)
ATTN: Paul Heckman
San Diego, CA 92152-5000

13. Naval Coastal System Center — 1
Navigation, Guidance, and Control Branch
ATTN: G. Dobeck
Panama City, FL 32407-5000

14. Hal Cook, Code u25 — 1
Naval Surface Warfare Center
White Oak, MD 20910

15. HQDA Artificial Intelligence Center — 1
ATTN: DACS-DMA, LTC A. Anconetoni
The Pentagon Room 1D659
Washington, DC 20310-0200

16. RADM G. Curtis, Code PMS-350 — 1
Naval Sea Systems Command
Washington, DC 20362-5101

17. Dr. David Y. Tseng — 1
Hughes Research Laboratories
3011 Malibu Canyon Rd.
Malibu, CA 90256

18. Research Administration 1
Code 012
Naval Postgraduate School
Monterey, CA 93943-5000

19. Russ Werneth 1
NASA Goddard Space Flight Center
Greenbelt Road
Greenbelt, MD 20771

Thesis
N853 Nordman
c.1 A computer simulation study of mission planning and control for the NPS Autonomous Underwater Vehicle.